IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR PATENT FOR

HIGH-AVAILABILITY SUPER SERVER

Inventors: Arthur C. McKinney
              162 Nale Drive
              Madison, AL 35758

              Charles H. McCarver, Jr.
              122 Cheyenne Trail
              Huntsville, AL 35806

              Vahid Samiee
              8617 Spicewood Springs Road
              Austin, TX 78759

Attorney Docket: 1247/A63

HIGH-AVAILABILITY SUPER SERVER

PRIORITY

5   This application is a continuation of U.S. Application No. 08/802,827, filed February 19, 1997, which claims priority from subject matter disclosed in two provisional applications entitled HIGH-AVAILABILITY SUPER SERVER, having serial number 60/011,979, filed February 20, 1996, and METHOD AND APPARATUS FOR SIGNAL HANDLING ON GTL-TYPE BUSES, having serial number 60/011,932,

10   filed February 20, 1996. Each of the above described applications are hereby incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to providing high-availability parallel processing super

15   servers.

SUMMARY

The present invention provides a high-availability parallel processing server that is a multi-processor computer with a segmented memory architecture. The processors are

20   grouped into processor clusters, with each cluster consisting of up to four processors in a preferred embodiment, and there may be up to 5 clusters of processors in a preferred embodiment. Each cluster of processors has dedicated memory buses for communicating with each of the memory segments. The invention is designed to be able to maintain coherent interaction between all processors and memory segments within a preferred

25   embodiment. A preferred embodiment uses Intel Pentium-Pro processors (hereinafter P6). The invention may be modified to utilize other processors, such as those produced by AMD or CYRIX. (Registered trademarks referenced herein belong to their respective owners.)

The present invention comprises a plurality of processor segments (a cluster of

30   one or more CPUs), memory segments (separate regions of memory), and memory communication buses (pathways to communicate with the memory segment). Each processor segment has a dedicated communication bus for interacting with each memory

segment, allowing different processors parallel access to different memory segments while working in parallel.

The processors, in a preferred embodiment, may further include an internal cache and flags associated with the cache to indicate when the data within the cache may be out of date, or if the data within the cache is data shared by other processors within the invention. Through use of setting the internal cache flag to a desired state, the contents of the internal cache of a processor may be effectively monitored from a vantage point external to the processor. This would allow for maintaining multi-processor cache coherency without requiring all processors to observe all other processors memory traffic.

The processors, in a preferred embodiment, may further comprise processors with caches external to the processor, where such cache may be an external write-back cache. There may also be caches associated with the memory communication buses to allow for enhanced memory access. In a preferred embodiment utilizing P6 processors, the processors may also be configured to operate in clusters of up to four processors to a processor segment. Note that this four processor limitation is one due to the P6 architecture. If an alternate processor is used, in which the alternate processor design allows more than four processors to a bus, then cluster size referenced herein may be adjusted accordingly.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic showing a high level bus structure of one embodiment of the invention.

FIG. 2 is a schematic showing a preferred maximum server configuration of one embodiment of the invention.

FIG. 3 is a schematic showing a preferred minimum desktop configuration, of one embodiment of the invention.

FIG. 4 is a schematic showing a XBus physical configuration according to the present invention.

FIG. 5 is a schematic showing a quad XBus processor segment of one embodiment of the invention.

FIG. 6 is a schematic showing a dual XBus processor segment of one embodiment of the invention.

FIG. 7 is a schematic showing a memory segment block diagram of one embodiment of the invention.

5   FIG. 8 is a schematic showing a memory segment layout of one embodiment of the invention.

Fig. 9 is a schematic of an exemplary XAP showing cache controller functionality.

10   **DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS**

The present invention provides a high-availability parallel processing server utilizing a scalable building-block architecture. Building-block design reduces development efforts due to block reuse in related architecture designs. A preferred embodiments of the invention may utilize the bus described in the provisional application

15   entitled "Method and Apparatus for Signal Handling on GTL-type Buses" (hereinafter "enhanced GTL-type bus"), serial no. 60/011,932, which was incorporated hereinabove by reference. In addition, a preferred embodiment may also utilize the timing technology disclosed in U.S. Patent no. 5,546,569 to Proebsting; this patent is incorporated herein by reference.

20   A preferred embodiment may have from one to twenty processors, although with the invention's present design, more are possible. A preferred embodiment will utilize P6 processors (hereinafter a preferred P6 embodiment), and thus up to four processors may be grouped to form a processor segment, and each processor segment is expected to have an internal bus rate of 66 to 75 MHZ. This bus rate is independent of the rate for the rest

25   of the computing system. Alternate embodiments utilizing non-P6 processors may have different internal bus rates, as the internal bus rate of a preferred embodiment is independent of the invention's communication protocols.

In a preferred P6 embodiment, each processor segment may also support up to four independent coherent buses, a limitation due to the P6 package and pin-connector

30   architecture; denser packaging technology may allow for more independent buses. Each coherent bus is separated into an address bus and a data bus. Preferred embodiments will utilize the enhanced GTL-type bus referenced hereinabove, so that each data bus may

operate at up to 4 GByte per second data transfer rate with four buses. In a preferred P6 embodiment, a sustained data transfer rate for each bus is expected to be approximately 710 MBytes per second. With utilization of the enhanced GTL bus, the data bus may operate at 133 to 150 MHZ, and the address bus posts 33 - 133 million transactions per

5   second, and may operate at 66 to 75 MHZ. Note that these operating speeds are possible if a different high-speed bus is used (e.g. ECL). In an alternate preferred embodiment having a standard non-enhanced (i.e. slow) bus, the data bus may operate at approximately 100 MHZ. The address bus may also be accordingly reduced in speed to compensate for the slower bus.

10   In a preferred embodiment, up to two I/O segments may be supported. Each Segment may provide up to eight PCI slots and four E/ISA slots, as well as include multiple SCSI, SSA, 10/100 MBit Ethernet, or Fiber Channels pathways. Preferably, up to four GBytes of main memory will be supported. An external tag/cache may be configured to support 256K or 512K cache lines. In addition, in a preferred embodiment,

15   different configurations of high availability may be chosen.

Preferred embodiments may also provide system administrators with information about recoverable system errors, faults, and operating environment, as well as to include programmable recovery actions in case of fatal errors. Preferably, a desktop management interface will allow the system administrator to monitor and configure the invention

20   locally or through a network.

Although the invention may be implemented with various central processing units (CPUs), in a preferred embodiment, the Intel P6 (or P6 MMX) processor will be used. According to the Intel specification, up to four P6 processors may be clustered on an In-Order split-transaction CPU bus to create a four way symmetrical multi-processor (SMP)

25   platform. As used in this specification and claims that follow, such clustering of CPUs is referenced as a "processor segment." Each processor / cache combination interfaces directly to the CPU bus, allowing system integrators to provide similar looking SMP hardware. Further detailed information regarding the P6 processor may be found the following Intel's references: *Pentium and Pentium Pro Processors and Related*

30   *Products*; and the *Pentium Pro Family Developer's Manual* three volume set: *Volume 1: Specifications* (Order Number 242690), *Volume 2: Programmer's Reference Manual* (Order Number 242691), and *Volume 3: Operating System Writer's Manual* (Order

Number 242692); these references are incorporated herein by reference. Additional information may also be found at Intel's World Wide Web site (http://www.intel.com).

FIG. 1 shows that a preferred P6 embodiment of the invention which extends the standard P6 quad-CPU SMP implementation to allow coherent integration of more than

5 one processor segment **100**. The P6 design architecture results in as many as twenty processors on five processor segments **102-108** being interconnected. This is accomplished through interconnecting processor **102-108**, input/output (I/O) **138**, and memory segments through multiple enhanced-GTL buses (hereinafter XBus) **110**. As noted hereinabove, more processors may be interconnected with alternate CPU

10 architectures. As shown, these multiple buses **110** form a subset configuration **136** (an XBus lattice) of a general cross-bar bus, where specific memory segments may be accessed through a dedicated bus.

In a preferred embodiment, the invention supports up to four memory segments **112-118**. Each non-memory segment interfaces to all memory segments through

15 dedicated address (XABus **140**) and data (XDBus **142**) bus-connection buses **120-134**. In a preferred embodiment, each processor segment **100-108** may access all memory segments **112-118** through different access buses, and different segments **100-108** may post simultaneous accesses to different memory segments without interfering with one another. However, processor segments attempting to access the same memory segment

20 must arbitrate for the bus leading to that memory segment. In this fashion, different memory segments remain highly available to different processor segments, and delayed access is reduced to occurrences of attempts to simultaneously access the same memory segment.

In a preferred embodiment of the invention, the XBus lattice **136** is a multiple bus

25 cross-bar supporting the requirements of high-end processors in a multi-processor environment. XBus agents will accommodate Intel P6 class processors and agents, and P6 agents will abide by the Intel P6 External Bus Specification without alterations. XBus agents will abide by the XBus specification. P6 bus signals will generate XBus signals and transactions. Thus, a preferred embodiment will coerce a blending of the XBus

30 protocol and P6 bus protocols.

A preferred embodiment also supports the MESI cache protocol, "read-and-invalidate" operations, and four sets of dual independent and cooperating buses **120-134**.

Each of the four sets **120-134**, in addition to the requisite control signals, comprise a 36

bit address (XABus) **120, 124, 128, 132** bus and a 72 bit data (XDBus) **122, 126, 130,**

134 bus. These buses support multiple outstanding transactions with out-of-order

response support, and they may operate fully synchronously and may run at the same

5 frequency as the CPU bus. A preferred embodiment will have the XABus **140** running at

a minimum frequency of 66 MHZ, and the XDBus **142** at a minimum of 133 MHZ. A

preferred embodiment operating at a preferred 133 MHZ supports source synchronous

transmission of data on XDBus **142**. An alternate preferred embodiment without source

synchronization, operating at slower speeds such as 66 to 100 MHZ, may instead

10 implement fewer XBuses and/or widen the data portion. For example, one such 100

MHZ embodiment may have only two XBuses and a data portion 144 bits wide. Data and

clock for sampling the data are delivered by the responding agent in order to minimize

transport delays and latency. This allows the data portion of the XDBus **142** to run at a

much higher clock rate, yielding substantially increased bandwidth. This configuration

15 allows for parallel distributed and overlapping arbitration, aggregate peak data transfer

rate of 4.2 GBytes per second (133 MHZ transfer rate), aggregate peak transaction

posting rate of 133 million per second, a low-voltage-swing open-drain GTL physical

layer for support of the XBus protocol, and a high operating frequency. In a preferred

embodiment, agents are also able to guarantee positive forward progress and avoid live-

20 lock situations when multiple agents cross access dirty cache lines in each others cache.

Preferably, error detection capability is available through support of parity on all buses,

and ECC memory with single bit correction and double bit detection should be

supported.

A preferred embodiment as described hereinabove thus has atomic transactions

25 *without* bus locking, supports use of semaphores, transient error recovery through

automatic retry of erroneous transactions (which preferably is logged and retrievable by

the hardware abstraction layer (a component of the NT operating system)), and critical

word first ordering.

FIG. 2 shows that in a preferred embodiment, the invention's architecture

30 comprises two pairs of ASICs. One pair implements an interface between the XBus **110**

and P6 buses (processor segment ASICs) while the other pair implements the memory

control functions (memory segment ASICs) interfacing the XBus **110** to the memory

arrays. The processor segment ASICs comprise a data path ASIC (XDP) **202** and an address path ASIC (XAP) **204**. The memory segment ASICs comprise am XMD **206** a data path ASIC and an XMC **208** combination address/memory control ASIC. A preferred embodiment may contain one, two, or four XBuses **110**, where each XBus has

5   an independent memory segment with XMD **206** and XMC **208** which interfaces to dedicated XDPs **202** and XAPs **204** on non-memory segments.

Shown in FIG. 2 is the quad-XBus configuration. Each XBus **110** supports plugging circuit boards containing a segment into a midplane for that bus, resulting in four possible sets of segments. In the FIG. 2 embodiment, for each of the XBuses, there

10   are three preferred segment types of segments that may be attached. The first segment type comprises the processor segment **216** having up to four P6 or equivalent processors **214**, four XAP **204** and XDP **202** ASICs, tag RAMs **210**, and four data caches **212** (with associated support logic). The second type comprises the I/O segment **218**, having up to two P6 or equivalent processors, up to two Intel 82454GX controllers **214** (Intel's PCI

15   bridge), four XAP **204** and XDP **202** ASICs, and tag RAMs **210** with their associated support logic. (Note that block **220** represents both the processor and I/O segments, and block **214** represents both the processors and Intel 82454GX controllers.) The third segments type comprises the memory segment **222**, having a single XMD **206** and XMC **208**, and up to one GByte of main memory **224** per segment. Depending on how memory

20   is to be configured, up to four each of the XMCs **208** and XMDs **206** may be utilized on each segment to accommodate interchanging memory segments.

XAP

Preferred embodiments of the XAP **204** architecture have several major functions, which are as follows. One function is a programmable stride. Stride is defined

25   as the number of contiguous cache lines that a XAP **204** will service. In effect, the stride acts as a modulo function for the entire memory address range. At boot time the XAP **204** responsible for a specific XBus **110** (FIG. 1) will be programmed with its stride. On single XBus **110** systems, the XAP **204** will be programmed to take responsibility for the entire address range, and therefore will have a stride of 4 GBytes. Systems with multiple

30   XBuses **110** will have their XAPs **204** programmed such that each XAP **204** is responsible for a small number of contiguous cache lines. In a preferred embodiment, such a programmable stride is for allowing the optimization of memory access strategy

with respect to a particular application environment. For example, in an environment having fine data set work, the XAPs **204** may be programmed for a stride of four cache lines. This will allow XAP-0 to launch transactions on XABus-0 for cache lines 0 through 3, XAP-1 to launch transactions on XABus-1 for cache lines 4 through 7, etc.

5 Such a configuration increases the probability that subsequent memory accesses would be to different memory segments, thus allowing the avoidance of memory precharge delays or stalling due to another access pending at the desired memory location. Similarly, in a database environment having larger and often fixed size data records, the stride may be configured to contain an entire data record. Thus, when record searching,

10 such a configuration increases the probability that subsequent memory accesses will be to different memory segments, once again allowing faster and somewhat overlapping memory accesses due to the potentially parallel nature of the memory configuration.

Another function is outgoing request queue management (XABus), which allows multiple simultaneous requests on an XABus **140** (FIG. 1). The XAP **204** will provide a

15 queue for storage and management of its pending transactions until XABus **140** becomes available. Transactions may be posted to XABus **140** according to the XBus protocol. The XAP **204** will request ownership of its XABus **140** upon detecting a P6 bus cycle providing an address. The stride will determine which XAP **204** will take ownership of the transaction. Transactions are posted to the XABus **140** two clock cycles after they are

20 posted to P6 bus if the XABus **140** is available and the outbound queue is empty. Alternatively, through an implementation of a mode switch, transactions may be delayed until the local segment's level 3 (L3) snoop result is delivered. This configuration would allow possible cancellation of the transaction prior to posting on the XABus **140**, thereby decreasing the XBus **110** load. In this case, if the XABus **140** is available, transactions

25 may be posted to the XABus **140** no later than three cycles after the address is asserted on the local P6 bus.

In a preferred embodiment utilizing the Intel P6 processor (or equivalent) and associated processor bus, the bus can have a maximum of 16 outstanding transactions (a maximum of 8 in-order and a maximum of 16 deferred). The XAP **204** will support these

30 transactions concurrently. According to Intel's implementation of the P6 processor, a single P6 cannot post more than four transactions to the P6 bus. Preferred embodiments,

whether utilizing the P6 or alternate processor, must support the maximum number of possible outstanding transactions.

In a preferred embodiment, the XAP **204** will become a cache line owner as soon as it posts an XABus **140** transaction for that cache line. This agent should provide appropriate snoop response if other agents access this cache line. This agent may use the pending request type and the potential cache line state in providing a snoop response. If it is determined that the cache line will be modified by the first agent, it must then be provided to the P6 bus of that agent. Later, the XAP **204** on the first segment may run an appropriate cycle on its local P6 bus based on the pending request from the second agent to retrieve the requested cache line. The cache line is then provided to the second agent. This method may be chained such that no more that one additional request to an identical address is tracked by an XAP **204**. Alternatively, the transaction from other agents may be retried on the XABus **140** by the agent that posted the first transaction. However, note that this action may degrade performance since the entire transaction on XBus **110** and P6 bus will be resubmitted.

Another function is outgoing request queue snooping on the XABus which allows the XAP **204** to monitor XABus **140** requests posted by other segments. That is, the XAP will provide snoop results to the XABus **140** for the cache lines in its request queue and the local segment's cache. The snoop response will be reflected on the XABus **140** by no more than two clocks after it receives the posted address. The XAP **204** communicates to the XDP **202** the necessary information for completing any transaction that has resulted in a dirty hit in its outgoing request queue.

Another function is incoming request servicing which allows the XAP **204** to monitor every request on the XABus **140**. The XAP **204** must service these requests when it detects that its segment has responsibility for the current XABus **140** posting. The XAP **204** will use the posted XABus **140** address to examine the contents of its local tag RAM **210**. The XAP **204** must be capable of servicing priority requests from other XBus **110** or P6 bus agents. Preferably such incoming requests will be serviced ahead of other queued requests. The target XAP **204** communicates to its companion XDP **202** the priority nature of the incoming request.

Another function is communication with the XDP **202**. The XAP **204** will provide information about its queued transaction to the XDP. The XDP **202** will use this

information to determine if it is the target for XDBus **142** (FIG. 1) transactions. The XDP **202** may also use this information to lookup contents of its external cache **212**. Contents of external cache are only used after determining that the contents are valid (with a tag RAM **210** hit). Queued transactions not posted to the XABus **140** will be removed from

5   the XAP's **204** outbound queue if the cache contents are valid. If the transaction is posted to the XABus **140**, then the XAP **204** must abort the ongoing memory access.

Another function is cache management (FIG. 9). The XAP **204** must store the status of its local P6 caches **901** in its external tag RAM **210**. The XAP **204** must manage contents of the cache **212** through the XDP **202**. In a preferred embodiment, the external

10   cache is a write back cache. The external cache **212** will contain, as a subset, a reflected copy of the local P6 cache contents. Preferably up to 64K cache lines are available in a single XBus **110** configuration. In addition, the cache **212** will include cache lines evicted by the local P6 processors. The XAP **204**, at a low priority and with available bus cycles on the local P6 bus, must determine if the status of the reflected cache lines are

15   correct (shared copies are in the P6 cache); this process is defined as cache cleansing. A problem that a preferred embodiment addresses is that no Intel specification for the P6 processor discloses how to make public the contents of the level 2 (L2) cache **901** serving a processor cluster. No method is disclosed for determining which internal cache line is being replaced when a new one is fetched. And, it may be inferred from Intel

20   disclosure that the Bus Read and Invalidate (BRIL) and Bus Invalidate Line (BIL) functions may or may not modify the requested line. That is, Intel disclosure indicates that the requests *intend* to modify the lines, but does not say they will; therefore the data associated with a requested line may still be maintained within a processor cache, and may not become stale unless the requested line is actually modified. Consequently, the

25   invention needs to forcibly track the L2 cache's contents. Towards this end, the XAP **204** tracks evicted P6 cache lines in producing a correct reflection of the P6 internal caches. If external TAGs indicate that a line is in the modified state, then XAP **204** must not include that line as part of the cleansing process. Invalidate or Read and Invalidate instructions issued by a P6 do not guarantee that the cache line will be modified by that

30   P6. External tag RAM **210** is managed by XAP. In a single processor segment configuration, the invention may function without tag RAMs. Such tracking of private

cache contents is not unique to the Intel P6 processor, and such tracking by the invention is applicable to any other processor that hides such information.

Preferred embodiments should allow false HITM support for the XBus. (The HITM is a cache buffer holding a cache line from an external inquiry that results in a
5   cache hit. Asserting HITM results in receiving cache writeback data.) It is possible, in a multiple segment system without a L3 cache, for the XAP **204** to assert HITM in response to an XABus **140** transaction. This would occur when a P6 is provided a cache line in the Exclusive state but that P6 has not modified the line or will not modify the line. This again is due to the lack of information regarding the state of internal cache
10   lines; thus one must assume that once a line is read with intent to modify, it must then be treated by the tag RAMs as potentially dirty and that a machine with speculative execution may prefetch the data and *not* modify it. In such a case the XAP's **204** snoop phase will result in no write-back data on the P6 bus. If a XAP **204** has determined that it asserted a HITM condition erroneously, it will resubmit the original transaction on the
15   XABus **140**. A preferred embodiment will have a predetermined address for doing so. The appropriate memory segment may determine from the new transaction that a false HITM condition exists and must therefore supply the cache line. The XAP **204** asserting the false HITM must set the tag for this cache line to "Invalid."

Note that a copy of modified cache lines that have not been evicted by local P6
20   processors may nonetheless be stale. Only evicted cache lines that have not been re-requested by the local P6 processors in an Exclusive state may be provided to other segments without running a Read, or Read and Invalidate transaction on the local P6 bus. A preferred embodiment will force P6 processors to store cache lines in a shared state. This in turn forces processors to post an Invalidate transaction if it requires to modify
25   shared cache lines. In this manner, the external tag RAM **210** / XAP **204** will be able to track modified lines. If lines were allowed to exist in an Exclusive state, then the P6 is allowed to modify the cache line without running any P6 bus cycles. This would force the XAP **204** to perform a Read transaction on its local segment to determine the true state of the cache line, a time consuming operation that a preferred embodiment of the invention
30   seeks to avoid.

The XAP **204** will post requests on the P6 bus as a priority agent for retrieving dirty cache lines requested by other segments. Dirty data must be returned to the

requesting agent on XDBus **142** via a high priority request. Request for returning dirty data must be pipelined with other transactions on the bus without unnecessary dead cycles. Responding segments that intend to provide write-back data must also provide an associated latency for providing that data. This latency is provided for dirty lines in both

5   L2 and L3 caches. The requesting segments may compare this latency to an internal approximation of when the dirty cache line is to be provided to their local P6 bus. Based on this comparison, the requesting segment may decide to defer the transaction or keep it in its in-order queue. Requesting segments that receive a snoop response indicating write-back from a remote processor segment's L2 cache may only keep such a

10   transaction in their in-order queue if they are not responsible for write-back data from their L2 due to an earlier transaction.

The status flags in the tag RAM **210** will allow XAP **204** to determine if it must respond to the posted transaction. P6 processors on a segment will be forced to store all cache lines according to MESI protocol unless they specifically execute a Read and

15   Invalidate or an Invalidate against that cache line. In this case the P6 posting the transaction will be allowed to store an Exclusive copy. Tag RAMs **210** must carry flags associated with the local P6 cache flags (MESI), as well as an ES and MF flag. The ES flag, for "exclusive to this segment," indicates that a cache line may be stored in a P6 cache as shared while the local XAP **204** identifies itself as the exclusive owner. The MF

20   flag, for "modified and flushed," indicates that the cache line that was stored by a P6 as Exclusive was later evicted. This also means that the local P6 segment has not re-requested this line.

Another function is write transactions to dirty lines. In a preferred embodiment, write-line transactions posted on XABus **140** with a write-back request that result in a

25   dirty hit (via HITM) must log an error signal for diagnostic purposes.

Another function is semaphore support. The XAP **204** must support locked transactions without locking out other XBus **110** agents from using the bus or locking the processor bus of remote segments. Local XAPs **204** must keep track of locked addresses. XBus **110** accesses intruding on a locked/guarded address will receive a RETRY

30   response to their posting on the XABus **140**. The XAP **204** must also insure forward progress and secure access to any P6 executing a locked transaction. Transactions against locked memory locations receive a RETRY response on the local P6 segment while the

XAP **204** accesses remote data. This allows other P6s on the issuing segment to proceed. The issuing XAP **204** will not allow access to locked cache lines by another XAP. Such accesses must be retried on the XBus **110** until the lock is removed. Multiple locked transactions may be pending as long as the locks are for different cache lines.

5   Agents with outstanding split locked transaction must assert RETRY on the XBus **110** if another agent attempts to access their locked cache line(s). Only cache lines that have successfully passed their snoop phase on the XABus **140** may be guarded by the XAP **204** issuing the split locked transaction (multiple XBuses are involved).

The agents with outstanding split locked transactions crossing page boundaries

10   must force a RETRY on the XBus **110** if another agent attempts to post a split locked transaction that crosses page boundaries. In a preferred embodiment, only a single split locked transaction crossing page boundaries is allowed on all XBuses within a system. Other agents will be able to post a new split locked transaction crossing page boundaries as soon as the snoop phase for the second read access of the posted locked transaction

15   (crossing page boundaries) is completed on XBus or the Lock on the issuing segment's bus is removed. Agents may post split locked or simple locked transactions on XBus while a split locked transaction crossing page boundaries is pending.

Another function is P6 delayed snoop cycle management. The XAP **204** will queue transactions from local P6 segments until they may be posted to XABus **140**. The

20   XAP **204** must manage and minimize snoop response delays on its P6 bus.

According to the Intel specification, the P6 bus requires a snoop results within four cycles after assertion of ADS on the P6 bus. In a multiple segment configuration, a HITM on XABus **140** will result in a deferral of the transaction on the issuing segment. In order to accomplish this, the issuing segment's snoop cycle must be delayed by 2

25   clocks to accommodate snoop results from XABus **140**. Only HITM on the XABus **140** will result in a P6 bus transaction deferral. In an embodiment having only a single segment, this feature is not necessary, since all snoops take place on the local P6 bus. In this limited configuration there will be no delay of the snoop phase. The agent asserting HITM will provide the data. Deferral and snoop phase delay features may be boot time

30   programmable. In a preferred embodiment, L3 cache hits will not cause snoop phase delays. Note that snoop phase may be delayed due to previous transactions that resulted with no L3 hit.

Another function is I/O read access deferral. Response to read accesses posted against I/O address space must be deferred by the local XAP **204**. Such deferrals will allow other CPUs on the issuing segment to proceed while the read data is being retrieved. The XAP **204** will provide for disabling deferrals for embodiments with a

5   single processor segment which and local PCI bridge.

Another function is minimized pipeline delays. The XAP **204** must minimize pipeline delays for posting transactions to the P6 bus or XABus **140**. Time critical information must be provided with no more than a single register delay. Each XAP **204** will preferably provide a complete 36 bit address bus on both XABus **140** and P6 bus. In

10   preferred embodiments of the invention, the XABus **140** input or output will operate at 66 to 75 MHZ using a synchronous clock. Recall, as discussed hereinabove, that the speed limitations are due to the limitations of the P6 architecture, and may not be applicable to other processor architectures. The P6 bus input or output will therefore preferably operate at 66 to 75 MHZ using the same synchronous clock. The XAP **204**

15   will may use round robin arbitration to acquire the XABus **140** or P6 bus. P6 bus arbitration will be based on P6 bus specifications. XABus **140** arbitration will be based on the following scheme: at reset, the agents will assume a predetermined priority based on their agent ID code. In a preferred embodiment of the invention, this will be mid-plane position-dependent valuation. After reset, the agent that wins mastery of the

20   XABus **140** will become the lowest priority. All remaining agents will move up in priority. An agent that has not arbitrated for the bus over a long period of time, when the remaining agents have become bus masters, will be the highest priority agent. The XAP **204** may keep ownership of the XABus **140** if no other agent is requesting ownership and it has outstanding transactions in its outgoing queue. Transaction posting on the XABus

25   **140** may consume two cycles. Except for arbitration signals, new agents must insure a dead cycle on the XABus **140** prior to asserting their signals.

Another function is I/O address mapping. The XAP **204** will service PCI bridges. It must therefore detect bus activity on the XBus **110** and the P6 bus associated with such bridges.

30   Another function is system Management Mode support. Agents generate SMI Acknowledgments on the XBus when a P6 CPU on their segment enter the System Management Mode handler. A second SMI Acknowledgment exits the processor from

the handler. The XAP **204** must ensure that only local P6 processors in System

Management Mode issue write transactions to the System Management Memory address

space. Access to any other portion of memory is unrestricted. The agent will indicate that

the current access is for the System Management address space (SMMEM). This will

5   insure that faulty processors will not disturb this address space. If System Management

Mode is not entered or exited properly and a write transaction is posted to XBus, then

that transaction must be aborted (ABORT) and the XBus **110** segment (XAP **204** / XDP

202) will submit a Hard Error Response to its P6 bus. If the request is queued and not

posted on the XBus, then it must be removed from the queue and a Hard Error Response

10   must be returned to its P6 bus. In a preferred embodiment, XAP **204** conforms to the

XBus protocol.

XDP

Preferred embodiments of the XDP **202** architecture have several major

functions, which are as follows. A first function is incoming data queue management.

15   The XAP **204** provides information to XDP **202** regarding transactions that it is expected

to retrieve from XDBus **142** (FIG. 1) for the local segment. The XDP **202** must retrieve

these transactions and post them to the local P6 segment. If immediate posting is not

possible, it must enter such responses into a queue until the correct time slot on P6 bus is

reached. The P6 bus requires in order response while XBus **110** provides out-of-order

20   response. The XDP **202** will manage transaction ordering between the two buses. The

XAP **204** will communicate with its companion XDP **202** concerning priority incoming

data. Such data will be provided on the P6 bus as a priority agent. This incoming priority

data will be placed ahead of any other in the incoming queue.

Another function is Data Cache management. The XDP **202** must retrieve cache

25   lines from the L3 data cache and present them to the local P6 segment if the XAP **204**

has determined that the cache entry is valid. The XDP **202** must also provide cache lines

to XDBus **142** under direction of XAP **204**.

Another function is write transactions to dirty lines. Write-byte transactions

posted on the XABus **140** that have resulted in a dirty hit (via HITM) cause both

30   segments, the requesting segment, and the one asserting the HITM, to provide their

respective cache lines on the XDBus **142**. Each segment must identify their cache line,

whether WPL or write-back, in their respective XDBus **142** transaction. In this case, the memory subsystem is responsible for combining the incoming cache lines.

Another function is minimized pipeline delays. The XDP **202** must minimize pipeline delays for posting transactions to the P6 bus or XDBus **142**. Time critical

5 information must be provided with no more than a single register delay.

Another function is outgoing data queue management. Transactions may be queued or posted to XDBus as soon as data is delivered from the local P6 segment. The XDP **202** will provide a low priority to cache eviction postings from its local segment or its external cache. Memory reference postings and dirty cache line write back due to

10 assertion of HITM by this segment must have a high priority. The XDP **202** must manage XBus utilization in a manner so as to not stall its local segment due to cache eviction. The XDP **202** will arbitrate for the XDBus **142** as a priority agent for priority outgoing cache lines as instructed by its companion XAP. Each XDP **202** provides a complete 72 bit data bus on both XDBus **142** and P6 bus. The XDBus **142** input or output will

15 preferably operate at 133 to 150 MHZ using a source synchronous clock. The P6 bus input or output will preferably operate at 66 to 75 MHZ. However, as discussed hereinabove, an alternate embodiment may operate at 100 MHZ (or slower) with an appropriate modification to the width of the data bus.

Another function is Arbitration and Priority. The XDP **202** may use round robin

20 arbitration to acquire the XDBus **142** or P6 bus. P6 bus arbitration will be based on the Pentium Pro bus specifications. In a preferred embodiment, an appropriate signal is asserted on the P6 bus, but arbitration between the XDP's is performed independent of the Pentium Pro bus protocol. XDBus **142** arbitration will be based on the following scheme: at reset, the agents will assume a predetermined priority based on their agent ID

25 code (mid plane position dependent). After rest, the agent that wins mastery of the XDBus **142** will become the lowest priority. All remaining agents will move up in priority. An agent that has not arbitrated for the bus over a long period of time, when the remaining agents have become bus masters, will be the highest priority agent. The XDP **202** may keep ownership of XDBus **142** if no other agent is requesting ownership and it

30 has outstanding transactions in its outgoing queue. Transaction posting on XDBus **142** may consume three cycles excluding the arbitration phase. New agents must insure a dead cycle on the XDBus **142** prior to asserting their signals (except arbitration signals).

Another function is transaction status. Each transaction must receive a response within a predetermined time period. This period is referred to as transaction time-out period and it must be programmable at boot time. The XMD **206** must assert report occurrence of an unrecoverable error (UERR) if a transaction does not receive a response

5   within the time-out period.

Another function is semaphore support. The XDP **202** will provide storage for at least two locked/guarded cache lines. The issuing P6 on the local segment will be allowed to proceed with a locked transaction only if the addressed cache line has been retrieved by the XDP **202**. The XDP **202** must inform XAP when guarded cache lines are

10   delivered. The XAP **204** will then allow the local P6 to proceed with its transaction by not retrying P6's request to post the locked transaction.

Another function is ECC support. The XDP **202** must be capable of detecting ECC errors on all data delivered or received from the XDBus **142**. The XDP **202** will effect its internal status register upon detecting multiple bit ECC errors. This information

15   will be used to isolate faulty portions of the system. Data delivered to The XDP **202** for transactions that are on top of its in order queue may not have sufficient time to be checked prior to delivery to P6 bus. In this case the XDP **202** may deliver the data and defer the ECC check until after delivery. Under such a condition, the P6 bus may receive a normal data response even though the delivered data has a multiple bit error. Deferred

20   checking of the delivered data will help isolate the faulty subsystem. The P6 will detect the ECC error due to its internal check mechanism. If the data is delivered to the XDP **202** from the XDBus **142** with a hard error response, then the same response is to be delivered to the P6 bus. All data queued by the XDP **202** for delivery to the P6 bus at a later time will have sufficient time for ECC checking. In this case a hard error response

25   must be delivered to the P6 bus when a multiple bit error is detected. In a preferred embodiment, the XDP **202** conforms to the XBus protocol.

XMC

Preferred embodiment, the XMC **208** has several major functions as follows. The first is incoming queue management. The controller must examine each address against

30   idle banks of memory. In an effort to keep all banks busy, accesses against idle banks must be given priority even though other requests may be in front of the queue. The XMC **208** will must support servicing of priority requests for read operations. The XMC

208 will give high priority to read operations. Write operations will be managed in a predetermined manner that will not cause XBus write stalls due to full queues.

Another function is cache line pre-fetch. Read requests for a complete cache line will result in the read of that cache line plus the read for the next cache line. The next

5 cache line is read in advance of an actual request only if a request is not pending against the bank containing the next cache line. The next cache line is placed in a line buffer in the XMD **206** awaiting potential arrival of a request for the second line. When a pre-fetched request for the cache line actually arrives, the XMC **208** will signal the XMD **206** to queue the line for posting onto the XDBus **142**. At the same time the next sequential

10 line will be pre-fetched from memory. In a preferred embodiment, the XMC **208** will provide a mechanism to re-use locations in the pre-fetch buffer that are occupied by old pre-fetched lines.

Another function is interleaving. The XMC **208** may simultaneously service as many memory banks as are present within a host computer architecture. In one preferred

15 embodiment, the XMC services four independent memory banks, each bank utilizing SDRAMs, which have an additional two or four banks (16 MBit / 64 MBit technology). This preferred configuration will provide as many as 16 (64 MBit technology) banks per XMC **208**. The XMC **208**, through a snoop capability, must be able to abort transactions to modified cache lines or to cache lines where data will be provided from the local

20 cache. Cache line pre-fetch will not take place if a cache line access is aborted. The XMC **208** must be capable of supporting snarf operations from XDBus **142** as a result of a dirty cache line hit.

Another function is XMD **206** support. In a preferred embodiment, the XMC **208** will communicate to the XMD **206** the status of ongoing transactions through a private

25 bi-directional bus.

Another function is write transactions to dirty lines. The XMC **208** must direct the XMD **206** to store (in a temporary buffer) dirty line data associated with a write-byte transaction. It must also direct the XMD **206** to combine the dirty line with the write-byte data when it arrives. The XMC **208**, after the lines have been combined, may then write

30 the resulting cache line into the memory. The memory is the owner of the cache line after the original write-byte transaction has been posted. The memory will not respond to a subsequent read request for that cache line until the write-byte data has been combined.

Another function is ECC support. The XMC **208** must support ECC operations provided in XMD **206**. In case of ECC, the corrected data must be written back into memory and provided on the XDBus **142**. The Data valid signal must not be asserted in case of an ECC error in the current line. The XMC **208** will support parity as provided

5   for XABus **140** control signals. XABus **140** transactions which include parity errors must result in a re-posting of the erroneous transaction.

Another function is self-snooping write queue. The XMC **208** must guarantee read data delivery from the write queue in case of an address match between the write queue and the read operation. Data should not be provided on XDBus **142** until the snoop

10   cycle (the pre-fetch) for a read transaction has been completed.

Another function is Memory Scrubbing. The XMC **208** will support memory scrubbing to insure data integrity. This may be implemented as a low priority task, and may be performed in conjunction with the memory refresh function. This function may require memory initialization at power up.

15   Another function is False HITM support. The XMC **208** must support the False HITM condition by changing the write back condition for a cache line to a read condition and providing the result through the XMD **206** on the XDBus **142**. The XMC **208** will receive indication of such a condition through a second posting of an original transaction on XABus **140**. Such postings will be made at a predetermined address. It is possible that

20   a False HITM may occur during a partial cache line write operation (WPL). In such cases, the XMC **208** will first cause a read of the cache line, allowing the XDP to combine it with WPL data when it arrives, and writing the result into its final destination. In a preferred embodiment, the XMC **208** will conform to XBus protocol.

XMD

25   Preferred embodiment, the XMC **208** has several major functions as follows. The first is write queue. The XMD **206** must contain a write queue capable of supporting a large number of transactions. The queue must be sufficiently large so as to not cause XBus write stalls. Write data will be written into main memory at a lower priority. Incoming data must be ECC checked prior to writing the data into memory. The write

30   queue must be self snooping, and the XMC **208** must guarantee read data delivery from the write queue in case of an address match between the write queue and the read operation.

Another function is read queue. The XMD **206** must support a read queue capable of supporting a large number of transactions. Up to four segments may be posting read requests to the XAP **204** while a single XMD **206** will respond to all posted transactions. Outgoing data must be ECC checked prior to assertion of data valid signal (DS). If an

5   error is detected, then such data may be provided on the bus, but data valid may not be asserted. For diagnostics purposes only, the XMD **206** will provide a mechanism that allows faulty data returns with assertion of a data valid signal. Priority requests on the XABus **140** should result in priority response on XDBus **142**. Such responses will must take priority in the front of the outgoing queue.

10   Another function is cache line pre-fetch buffer. The XMD **206** must support a buffer for supporting a large number of pre-fetched cache lines. The XMC **208** will indicate to the XMD **206** which pre-fetched buffer location should be queued for posting.

Another function is write transactions to dirty lines. Write-byte transactions posted on the XABus **140** that have resulted in a dirty hit (via HITM) cause both

15   segments, the requesting segment and the one asserting HITM, to provide their respective cache lines on XDBus **142**. Each segment must identify their cache line (WPL or write-back) in their respective XDBus **142** transaction. In this case the memory subsystem is responsible for combining the incoming cache lines. The memory is the owner of the cache line after the original write-byte transaction has been posted. The memory must not

20   respond to a subsequent read request for that cache line until the write-byte data has been combined. In case of a false HITM, the memory subsystem will combine the WPL data with the addressed cache line.

Another function is XMC **208** support. In a preferred embodiment, the XMD **206** will communicate with the XMC **208** regarding ongoing transactions.

25   Another function is that the XMD **206** must become a priority agent on XDBus **142** for returning read data if its return buffer is nearly full. This will momentarily force posted writes on XDBus **142** to become low priority and reduce read return latency. The XMD **206** will use round robin arbitration to acquire the XDBus **142** based on the following scheme: at reset, the agents will assume a predetermined priority based on their

30   agent ID code. In preferred embodiments of the invention, this will be mid plane position dependent valuation. After rest, the agent that wins mastery of the XDBus **142** will become the lowest priority. All remaining agents will move up in priority. An agent that

has not arbitrated for the bus over a long period of time, when the remaining agents have become bus masters, will be the highest priority agent. The XDP may keep ownership of the XDBus **142** if no other agent is requesting ownership and it has outstanding transactions in its outgoing queue. Transaction posting on the XDBus **142** will consume

5   three cycles excluding the arbitration phase. Except for arbitration signals, new agents must ensure a dead cycle on the XDBus **142** prior to asserting their signals. This arbitration method shows all device requests to all device on the bus, and all devices then simultaneously apply (i.e. in parallel) the same arbitration method to determine if it won the arbitration contest. In this fashion all devices know arbitration results without extra

10   time taken to notify them.

Another function is false HITM support. The XMD **206** must support False HITM condition as required by its companion XMC **208**. See also hereinabove regarding False HITM.

Another function is ECC support. The XMD **206** is responsible for ECC checking

15   all data it delivers or receives to/from the XDBus **142**. The XMD **206** must set internal status registers and record the error. When delivering data to the XDBus **142**, the XMD **206** may not have sufficient time to provide the correct response type (normal/hard error) if a multiple bit ECC is detected in the last 16 bytes of the cache line and the cache line is required for immediate delivery to XDBus **142**. In this case XMD **206** will continue to

20   deliver the cache line. The XMD **206** is responsible for indicating a hard error response on the XDBus **142** if a multiple bit error is detected in the first 16 bytes of data that is to be delivered to the XDBus **142**. The XMD **206** is also responsible for providing a hard error response on the XDBus **142** for all cache lines that contain a multiple bit ECC error and are queued for delivery at a later time.

25   **Tagging and Caching**

Intel's P6 processors contain four way set associative cache subsystems. A fundamental problem that a preferred embodiment addresses is that the cache controller does not provide external information regarding the cache lines it is replacing. An external device can attempt to run snoop cycles on the P6 bus to determine which cache

30   lines are being replaced, but this is very inefficient and requires creating a large number of transactions on a P6 bus just for this purpose.

In order to minimize snoop cycles on the P6 bus, while also maintaining coherency in preferred embodiments, the following facilities are provided by the tag controller **905**, as shown in figure 9. The first is that a large external tag be able to accommodate up to 16 to 32 times the number of cache lines that may be stored by a

5 single quad-CPU processor segment.

Another is that the CPUs will not be allowed to store cache lines in the Exclusive state unless they specifically execute an Invalidate or Read and Invalidate cycle. As long as lines are stored in a shared state, there will be no immediate need to run snoop cycles on the P6 bus to determine the actual state of the line being requested by another

10 Segment.

Another is that local TAGs store sufficient information to indicate if a segment is an Exclusive owner of a cache line, even if the CPU is forced to store the cache line in the Shared state. The tag controller **905** will monitor write cycles initiated by the P6 to determine which dirty cache lines are being evicted.

15 Another is that the tag controller **905** will declare ownership of cache lines via assertion of HITM on the XABus **140** that a P6 on another segment has requested. If such declaration is made, then the tag controller will post an appropriate cycle on the local P6 segment (or preferably retrieve the line from the external cache if it has been previously evicted) and provide to the XBus. Another is that the tag controller **905** will perform

20 snooper **902** function which sequentially scans the tag RAM for shared cache lines and, at a low priority, when P6 bus is idle, will run snoop cycles to determine if the line is currently present (tag cleansing). Modified lines will not be snooped for cleansing purposes. Preferably, external tag depth will be limited to 256K or 512K cache lines per XBus **110** (FIG. 1) connection. Systems with a single processor segment will be able to

25 function without external tag.

An external cache **212** may be added to remedy two known problems. The first is that local processors are not guaranteed to evenly distribute transactions among XBuses within a system. (This is not a problem unique to Intel processors.) Processor transactions may target the same bus repeatedly causing bus congestion on one XBus

30 while another is predominantly free. Responses to transactions posted against nearly idle buses will be provided sooner than transactions posted against congested buses. Due to the in-order requirement, such early responses may not be used until the response to the

previously posted transactions are provided. In addition one must consider that multiple segments will be posting transactions against all XBuses. This will cause further delay in responses to this segment. Such a condition will force processor stalls due to lack of data.

The second is that processor caches may be relatively small (with respect to data
5   set size and application being executed), making it very likely that cache lines are frequently replaced. The invention's P6s will have to compete with large number of other CPUs for memory references. This may delay delivery of data and cause processor stall.

Through using a local cache, some responses will be provided locally and faster than retrieval from main memory. For P6 processors, all transactions are always posted to
10 the XBus 110 unless the posting is delayed through queue management in the outbound queue, and the XAP 204 determines that the local cache contents are valid. Local cache responses will allow termination of posted memory references thereby off loading the XBus for servicing other processors. Although TAG ram is required for maintaining cache coherency as described hereinabove, the external cache ram is an option for
15 improving system performance. In preferred embodiments of the invention, external cache contents will be inclusive of L2/L1 contents. External cache depth per XBus connection may be limited to 8 to 16 MBytes per segment, or 32 to 64 MBytes per segment in a fully configured FIG. 2 type of system.

FIG. 3 shows a preferred minimum desktop configuration. Such a configuration
20 would contain one of each type of ASIC. That is, in this embodiment, there is a processor segment 300, a single XBus 110 (shown on FIG. 1) for address 308 and data 310, and a single memory segment 302. I/O capability may be added to this embodiment by adding Intel 82454GX controllers 304 directly to the P6 bus. This embodiment is expected to have at most four P6 processors and two PCI bridges, with support of up to one GByte of
25 memory 302. A somewhat more advanced configuration could be achieved by providing two XBuses 110 and eight to ten processors 306 along with an I/O Segment (not shown).

FIG. 4 shows that in a preferred embodiment, the XBus 110 (shown in FIG. 1) is expected to be implemented on a two sided midplane 400, where installed within the computer are two memory segments 402, five processor segments 404, 412, and one I/O
30 Segment 406. In an alternate embodiment, a second I/O processor may be substituted for processor 412. In FIG. 4, each memory segment 402 services two XBuses 408, 410. Each board connects to the midplane with several low inductance connectors. During

operation of the invention, a P6 CPU on a processor segment will post a transaction to an
XBus **408, 410** as soon as it acquires the local P6 bus and starts the P6 bus Request
Phase. This action is monitored by all XAPs **204** (FIG. 2) on that segment. One of the
XAPs **204** will route this transaction to an XABus **140** (FIG. 1). Since a system may have
5   one or more XBuses, at boot time, each XAP **204** and XMC **208** (FIG. 2) will be
configured to respond to certain addresses.

XAPs **204** / XMCs **208** on the same bus but belonging to other segments are
continually monitoring XABuses **140** (FIG. 1) for transactions. Target segment(s) will
initiate appropriate action on their segment in order to respond to the posted XABus **140**
10   request. The target segment, when the response is available, will post the response to
XDBus **142** (FIG. 1) through its XDP **202** / XMD **206**. The XDPs/XMDs for all
segments are continually monitoring XDBuses **142** for responses that may belong to their
local segment. The XDP/XMD will provide such a response to the local P6 bus/memory
for that segment. The XAP **204** / XMC **208** communicates launched transaction
15   information to the local segment's XDP **202** / XMD **206**. This will include critical
information such as transaction ID, expected response, portions of address needed for
concurrent local external cache look-up, etc. This will enable XDPs/XMD to properly
route XDBus **142** postings. If a valid cache line is currently stored in the cache, a bus'
data cache **212** (FIG. 2) may intervene on behalf of the memory and locally provide data.

20   **Board Layout**

The following sections illustrate board layouts that may be used in high-end and
mid size servers. In some embodiments of the invention, the XBus and its associated
ASICs may reside on the mother board, a no mid-plane design. Such a configuration may
reduce packaging size for these products. Embodiments configured with mother boards
25   may cost more, and force customers to purchase more complete products up front, as all
ASICs will be on the mother board and the P6 CPUs will become the main option. The
invention's mid-plane based products will allow customers to purchase a minimum
chassis and then configure their product over time.

FIG. 5 shows the processor segments in a preferred embodiment of the four XBus
30   system configuration. This embodiment has SMT straddle mounted connectors for
XABus **140** and XDBus **142**. Immediately above each connector will be a data path **202**
or address path **204** ASIC. Preferably the connection length between the ASIC and the

connectors is minimized, and preferred embodiments will utilize stub connections. Further, each side of the ASIC must be used for a specific interface: XBus (XDP **202** / XAP **204**), tag **210** or cache **212**, P6 bus, and XAP **204** or XDP **204** interconnect.

5   The tag RAM **210** is fast access, preferably 5-8 n sec, static RAMs mounted on a SIMM or its equivalent (e.g. DIMM). The SIMM will house as many as twelve RAMs for tracking 512 K cache lines. A preferred minimum tag configuration will contain six RAMs on one side of the SIMM for tracking 256 K cache lines. The tag RAMs are not optional and one of the two possible options must be resident. In order to enable a high speed design, each SIMM will service only one XAP. A more complex embodiment may

10   be configured with 1,024K cache lines per XBus, through use of higher density RAM technology.

In a preferred embodiment, the L3 Cache **212** will be designed around SDRAM technology to allow fast cache burst access. The SDRAMs will be mounted on a SIMM and provide a 72 bit data path. Due to available memory size, the SIMMs may contain as

15   much as 1,024K cache lines. TAGs presently track only up to 512K cache lines. As TAGs are a critical component of the system, every effort is made to minimize cache line access time.

Each processor segment will preferably house up to four CPUs **500** and associated power conditioning circuitry **502**. Although preferred embodiments of the

20   invention will utilize CPUs with a low profile heat sink that extends beyond the CPUs, other embodiments may use heat pipes, flanges, fanned enclosures, or other cooling devices.

FIG. 6 shows an embodiment having dual XBus (XDP **202** / XAP **204**)segments. These segments are de-populated versions of the quad XBus segments shown in FIG. 5.

25   **I/O Segment**

In a preferred embodiment, I/O and processor segments are very similar in layout except two of the P6s and associated power conditioners, preferably located at the rear of the board, are replaced by Intel's 82454GX PCI Bridge. I/O Segments will be available with dual or quad XBuses. Additional logic will provide VGA, 100 MBit Ethernet, and

30   P2P (PCI to PCI bridge with integrated I960 processor) capability. P2P may be used for multiple channel RAID and communication capability. Interconnection of I/O boards

(preferably PCI or E/ISA) with the core is mainly a mechanical issue but one option was shown hereinabove for FIG. 4.

Memory Architecture

FIG. 7 shows a memory bank of a memory segment. The memory segment in a

5   preferred embodiment may be constructed using SDRAM (or its equivalent) memory technology. Use of SDRAMs, however, will allow data retrieval at bus clock rates without any need for additional logic (ASICs) for interleaving the memory structure. This minimizes design complexity and reduces the need for precious real estate, while also reducing latency, improving performance, and allowing smaller memory configurations

10   while supplying multiple memory banks. Each memory segment, in addition to SDRAMs, will include an XMC **208**, an XMD **206**, and up to one GByte of memory in four independent memory banks **700**.

Each bank **700**, as shown in FIG. 7, is connected with a common 144 bit data path on the MDBus **702** to the XMD **206**. Each bank will have separate address and

15   control allowing simultaneous accesses over the MABus **704**. A MCBus **706** control bus is provided for allowing communication between the XMC **208** and the XMD **206**. In a preferred embodiment, the XMC **208** maximizes throughput on the MDBus **702** through management of its internal queues. In turn, the XMD **206** maximizes throughput to the XDBus **142** through management of its internal queues. The clock rate on the MDBus

20   **702** will preferably be 66 to 75 MHZ (144 bit transfer) while the clock rate on the XDP will preferably be 133 to 150 MHZ (presuming 72 bit transfers). See discussion hereinabove regarding slower bus speeds and greater data path width.

In preferred embodiments of the invention, a maximum of 33 million accesses per second may be posted to a memory segment, which is a reference for every other clock

25   from the same processor/I/O Segment. This may result in a maximum of 33 million data responses from one memory segment (over 1 GByte/sec). This large access capacity will allow a memory segment to support every access made by each CPU, as well as simultaneous accesses that may be directed at the same memory segment by multiple processor segments. The memory segment may also support the pre-fetch of the next

30   cache line before it is requested, aborted references to dirty cache lines that will be provided by other processor segments, and aborted references due to hits is external cache.

FIG. 8 shows one possible layout for the memory board. In this design two SIMM
sets are provided. Each set can house one GByte (Bank 01, or Bank 23). One set will be
used for XBus_0 **800** and XBus_1 **802** while the other will be used for XBus_2 **804** and
XBus_3 **806**. In a preferred embodiment, only one set will be populated. This

5   configuration will help accommodate the high data transfer rates from the SIMMs to
their associated XMDs **206**.